Q4)Describe how functions are working.

Optimization Analyzer project are working:

Constant Folding and Propagation:

Step 1: The input C# code is provided to the analyzer.

Step 2: The analyzer identifies constant expressions within the code, such as "int result = 5 + 3;" where "5 + 3" is a constant expression.

Step 3: The constant expression is evaluated at compile time to obtain its computed value, which is then propagated to replace the original expression in the code.

Step 4: The optimized code is generated, with the constant expressions replaced by their computed values, improving the program's efficiency by minimizing the need for runtime computations.

Dead Code Elimination:

Step 1: The input C# code is provided to the analyzer.

Step 2: The analyzer examines each line of the code to identify any parts that are redundant, unused, or do not contribute to the program's output.

Step 3: Code segments that are identified as dead code, such as empty or whitespace expressions, or variables that are never used, are eliminated from the code.

Step 4: The optimized code is generated, with the dead code removed, resulting in a more streamlined and efficient program.

Strength reduction :

Step 1: Identify repetitive or expensive arithmetic operations, such as multiplications or divisions by powers of 2.

Step 2: Replace these operations with equivalent, less expensive operations, such as bit shifts or additions, whenever possible.

Step 3: For example, replacing a multiplication by 8 with a left shift by 3, as shifting bits is generally faster and more efficient than performing a multiplication operation.

Decreasing Complexity:

Step 1: Identify complex arithmetic expressions that involve expensive operations or large numbers.

Step 2: Simplify these expressions to reduce the overall complexity and cost of computation.

Step 3: For instance, replacing "3 * 8" with "24" simplifies the expression, reducing the computational overhead.

aas